AVCore编解码重构 - 代码简介

1.1 重构背景

随着avcore编解码器的增多以及跨平台的支持，原有的编解码架构变得越来越难以维护，主要表现在以下几个方面：
1. 增加新的编解码器变得繁琐，每一个编解码器都存在代码冗余
2. 由于大量宏开关包括跨平台宏、编解码器开关宏、特性宏的使用，代码难以维护
3. 新的编解码能力尤其是具备硬件加速的编解码，很难融入现有的架构，新功能开发困难，代码常伴以修修补补，进而引起代码质量难以控制。
4. 无法对外输出统一的编解码能力及调用接口，跨模块重用困难

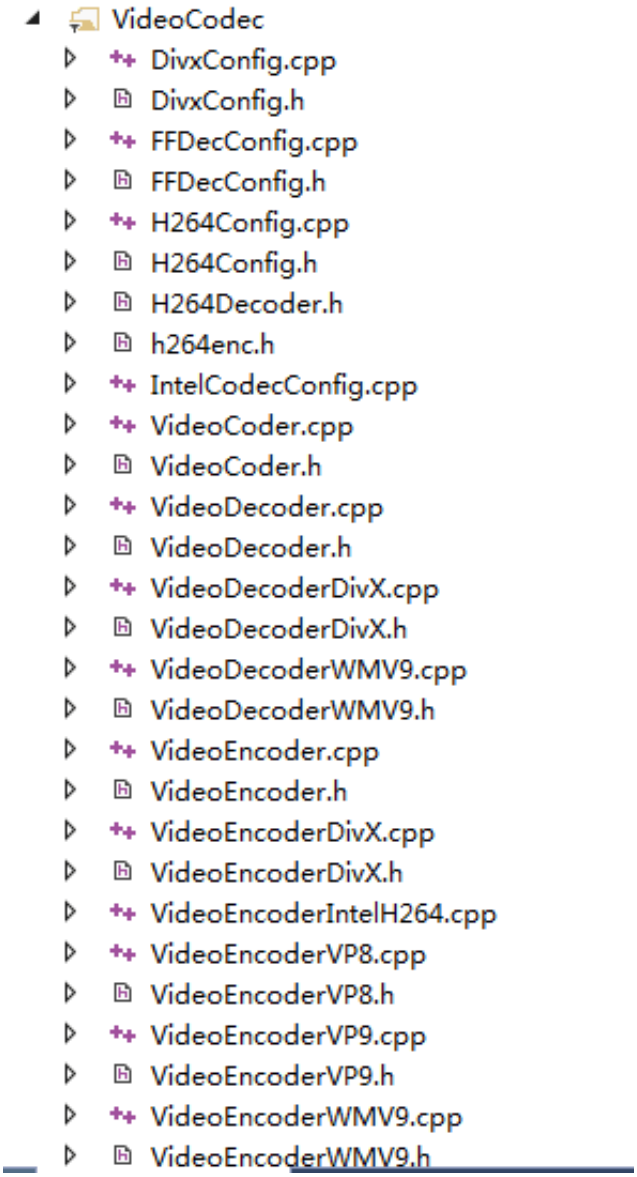
1.2 重构思想

基于之前维护avcore编解码模块的经验及开发新特性时带来的困扰，新的重构除了解决存在的问题以外，也应具备适应未来需求的良好扩展性，具有以下几个主要特点：
1. avcore启动时的动态编解码能力探测，并输出到编解码中心，每个编解码模块具备一系列属性，如是否具有硬件加速能力，是否具备直接渲染能力，相应的权重值等。如果相应的模块不存在，则不会在编解码能力中心输出，避免调用编解码器失败。
2. 统一每个codec的调用接口，整合软硬件编解码器，以统一的方式查询调用，具体的实现细节调用者无需关心，只需实现接口即可。
3. 精简codec调用流程，加入新codec变得异常简单，只需实现既定的接口即可。
4. 抛弃大部分宏使用，将平台相关的能力输出尽量隐藏到具体的codec实现中，简化核心层代码，增加可维护性与健壮性。
5. 简化了软硬解码自动切换流程，更方面的实现硬编解路数控制。
6. 可以方便实现codec失败回退功能，如有需要，未来可实现动态加载codec。
7. 统一的接口来实现向decoder/encoder传递参数。

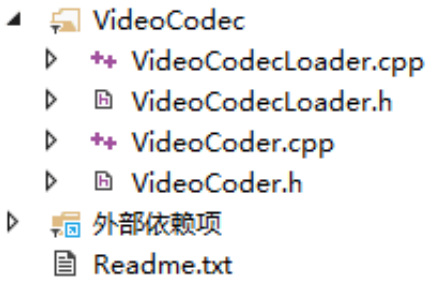
以简单的代码与逻辑实现以上功能，遵循"simple is the best"的设计思想。

1.2.1 重构前后对比

先来两张图直观看下重构前后的代码结构对比。
重构前的videoCodec目录：



重构后的videoCodec目录：



重构前如何启动一个解码器：

```
//----------------------
HANDLE  VIDEO_Decode_StartDecompress( int nCodeID,const BITMAPINFOHEADER &biOut )
{
    CVideoDecoder *pDecoder = NULL;
    LOG("INF:VIDEO_Decode_StartDecompress CodeID[%d].\n", nCodeID);

    switch( nCodeID )
    {
#if USE_FFWMV9DEC
    case VIDEO_CODEC_WMV9:
        LOG(_T("INF:VIDEO_Decode_StartDecompress create CVideoDecoderFFWMV9!\n"));
        pDecoder = new CVideoDecoderFFWMV9;
        break;
#endif
#if USE_DIVXCODEC
    case VIDEO_CODEC_DIVX:
        LOG("INF:VIDEO_Decode_StartDecompress create CVideoDecoderDivX!\n");
        pDecoder = new CVideoDecoderDivX;
        break;
#endif
#if USE_MSWMV9DEC
    case VIDEO_CODEC_WMV9:
        LOG("INF:VIDEO_Decode_StartDecompress create CVideoDecoderWMV9!\n");
        pDecoder = new CVideoDecoderWMV9;
        break;
#endif
#if USE_XVIDCODEC
    case VIDEO_CODEC_XVID:
        LOG("INF:VIDEO_Decode_StartDecompress create CVideoDecoderXvid!\n");
        pDecoder = new CVideoDecoderXvid;
        break;
#endif
#if USE_VPXVP8DEC//@@@fixme ÕâÀïÔõÃ´Ã»´ò¿ª£¬µ¼ÖÂ¿´androidºÚÆÁ
    case VIDEO_CODEC_VP8:
```

重构后如何启动一个解码器：

```
VCodecHandle VIDEO_Decode_StartDecompress(int nCodecId, const BITMAPINFOHEADER &biOut)
{
    VideoCodecDllEx *pCodecDll;
    HANDLE hCodec;
    VCodecHandle vHandle;
    VideoCodecPluginInfoEx vInfo;
    BOOL bHasHW = FALSE;
    BOOL isHW = FALSE;
    vHandle.pCodec = NULL;
    vHandle.pCodecDll = NULL;

    //if (nCodecId == VIDEO_CODEC_H264 || nCodecId == VIDEO_CODEC_H264PLUS)
       // nCodecId = VIDEO_CODEC_H265;

    if (!g_videoCodecLoader.IsInited())
        return vHandle;

    //check HW acc control
    bHasHW = g_videoCodecLoader.HasHWAcc(nCodecId, FALSE);
    if (bHasHW && g_hwAccController.IncreaseCurHWAccDecoder()) {
        isHW = TRUE;
    }

    pCodecDll = g_videoCodecLoader.FindCodecById(nCodecId, FALSE, isHW);
    if (!pCodecDll)
        return vHandle;

    pCodecDll->DllGetInfo(&vInfo);
    if (vInfo.bIsHWAccel) {
        vHandle.pCodecDll = pCodecDll;
#ifdef _FS_OS_ANDROID
        pCodecDll->DllDecConfig(NULL, DEC_CONFIG_TYPE_JVM, g_hVideoModule);
#endif
        return vHandle;
    }

    hCodec = pCodecDll->DllDecOpen(&biOut);
    if (!hCodec)
        return vHandle;

    vHandle.pCodec = hCodec;
    vHandle.pCodecDll = (HANDLE)pCodecDll;

```

以上仅从两个方面反映了重构前后codec创建使用的对比，无论从代码精简度、调用方便性还是功能健壮性上，重构后都得到了很大提升。

1.3 代码模块介绍

由于重构实现思路相对简单，模块化强，此处没有给出相应的流程，从代码实现中可以直接体现。
下面列出评审代码时的主要代码文件及相应的功能：

videoCodecLoader.cpp：
实现了codec的初始化，输出到codec中心，将avcore实现的codec模块化到相应的dll中，比如intel decoder dll， x264 encoder dll，ffmpeg decoder dll，调用系统相关接口实现硬件加速功能

的codec及无源码的第三方dll，全部放在vcodecWrapper的dll中。
同时也实现了根据codec id、是否需要硬件加速、解码还是编码等参数来查找codec的函数，供更上一层调用，是对外的接口。

videoCoder.cpp：
实现了重构后的编解码器调用，软/硬编解切换，硬编解路数控制。

render_proxy_coded_video.cpp:
实现了重构后的解码渲染流程，展示了如何统一硬件加速与软解的解码渲染流程。

videoCodecPlugin.h:
增加了统一的dll调用接口。

intelMediaSDKDecoder.cpp:
展示了如何实现dll调用接口。

作为对比可以参考重构前目录中相应的实现，了解重构对编解码模块带来的优势。

1.4 自我评价

以简单的思想实现重构，基本上实现了重构目标，但还有一些细节并未实现，比如屏幕共享有需求只硬解不直接渲染，动态codec加载虽在设计时预留了实现渠道但并未实现，后续可根据需要实现更多功能。

从代码规范上，存在一些函数过长的问题，但由于属于众多模块初始化过程，代码简单，不会影响代码理解，所以并未做完全的规范化。